Characteristics of Long-term Learning in Soar and its Application to the Utility Problem

William G. Kennedy BILL.KENNEDY@HQ.DOE.GOV
George Mason University, 9812 Ceralene Drive, Fairfax, VA 22032 USA

Kenneth A. De Jong KDEJONG@GMU.EDU
George Mason University, 4400 University Drive, Fairfax, VA 22030 USA

Abstract

Much of the work in machine learning has focused on demonstrating the efficacy of learning techniques using training and testing phases. On-line learning over the long term places different demands on symbolic machine learning techniques and raises a different set of questions for symbolic learning than for empirical learning. We have instrumented Soar to collect data and characterize the long-term learning behavior of Soar and demonstrate an effective approach to the utility problem. In this paper we describe our approach and provide results.

1. Introduction

It is generally accepted that important advances in artificial intelligence will require significant amounts of knowledge. Machine learning techniques have been developed to help automate the process of acquiring such knowledge. Much of the work in machine learning has focused on situations in which there are distinct training and testing phases. However, Allen Newell stated: "I believed ... learning was something that went on continuously and not at the volition of the agent" (Newell 1993). The recent increase in interest in agent-based systems involving dynamically changing environments has led to an increased attention to "on-line learning", i.e., systems in which learning mechanisms are always active.

Although on-line empirical learning has been studied in the machine learning community, running a symbolic learning system until it reaches steady state has not. Running a symbolic learner continuously over the long term places different demands on the learning techniques and raises a different set of questions than it does for empirical learning. A desire to understand the implications of running a symbolic learner long enough to see steady-state behavior led to exploring long-term learning in Soar.

Soar is a system developed by John Laird, Paul Rosenbloom, and Allen Newell (1986) that tightly couples problem solving and learning. In Soar, long-term learning is accomplished through solving a long series of problems. Soar is an established learning and problem-solving system based on formulating all tasks as searches of state spaces and using chunking as its only learning mechanism.

Soar's basic operation is to decide which operator to apply to the current state to transform it into another state. To decide, Soar fires all rules whose conditions are satisfied. If the result is that Soar has a single operator it can apply to the current state, it does so. When Soar does not have enough knowledge immediately available to decide on the next step, it creates a subgoal to resolve the "impasse." Soar learns by converting the results of successful subgoal problem-solving into a "chunk" retaining only the initial conditions and results. The chunk is then added to the knowledge base as another rule and is retained indefinitely.

Machine learning is based on the expectation of improved performance due to learning. However, the utility problem identified by Steve Minton (1988) signaled, in the long term, a fundamental, dibilating growth in problem-solving time with the number of rules in the system. His system, and others, demonstrated degraded performance in problem-solving time in as little as 9, 20, 52, or 100 problems (Bostrom 1992; Iba 1989; Markovitch & Scott 1988; Mooney 1989; Minton 1988, and Tambe, Newell, & Rosenbloom, 1990). That degraded performance has been addressed for only very small numbers of problems when compared to what would be expected in on-line learning. So, is long-term symbolic learning doomed by the utility problem?

Ten years ago, Bob Doorenbos and his collaborators provided the first impressions of long-term learning in Soar (Doorenbos, Tambe & Newell, 1992; Doorenbos 1993). His Dispatcher-Soar system was run for 6,550 problems. His focus was improvements in time to match the current state to applicable rules. However, he did not

Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

eliminate the utility problem because it does not address the increased branching factor caused by the additional rules. However, his foray invited others to look into long-term learning.

Fundamental questions for long-term learning research include whether learning will go on forever and how much of learned knowledge is used. To understand these issues better, we have instrumented Soar to let us collect data and characterize its long-term behavior. Then, based on an understanding of these characteristics, we modified Soar to address the utility problem. Here we describe our approach and results.

2. Approach

Our focus has been to understand better the answers to the fundamental questions regarding the behavior of Soar when run for extended periods of time in on-line learning mode. To address these questions, we reviewed previous work in long-term symbolic learning, we re-analyzed previous experimental data when appropriate, and we conducted new experiments using Soar to provide additional insights. In particular, we are indebted to Bob Doorenbos who has made available some of his trace data from Dispatcher-Soar experiments and provided assistance in interpreting the data.

The new experiments were conducted using a simple physics problem domain, the familiar Blocks-World domain, a concept acquisition system, and a tank warfare simulation. All of these systems were implemented in Soar and were available from the Soar Group (http://ai.eecs.umich.edu/soar/).

The physics problems were of the form: given an initial acceleration and velocity, what is the average velocity and distance covered after a given duration. The Blocks-World problems were to identify the moves necessary to change one configuration of a set of N named blocks into another configuration. The Blocks-World domain was chosen because it was expected to scale up nicely from small problem spaces to large but finite spaces as the number of blocks is increased. A Symbolic Concept Acquisition (SCA) system developed by Craig Miller (Miller 1993) was exercised with 500 training examples and 1,000 testing examples drawn from the chess end-game data (Blake, Keogh, & Merz 1998). The tank warfare simulation environment, TankSoar, is also available from the Soar Group. Here a problem was defined as deciding what action the tank should take next.

For these new experiments, Soar was instrumented to collect various kinds of internal data. Long-term behavioral data was then obtained by running Soar in an on-line learning mode and presenting it with long sequences of randomly selected problem instances (e.g., 10,000). We report our findings in the following sections.

3. Will Learning Go on Forever?

If we have a system that is capable of on-line learning, it seems inevitable that it would eventually learn enough new knowledge to suffer from the performance degradation of the utility problem. However, that belief is based on the expectation that learning continues indefinitely as Newell (1990) believed. We found Soar does NOT continue learning indefinitely.

3.1 Previous Research

Only a few researchers have run symbolic learning systems on long series of problems. The runs that made Soar famous by correlating its learning with data on human learning (Rosenbloom & Newell 1986) were 268 and 259 trials, effectively problems.

The first long-term learning experiments were done by Bob Doorenbos and his collaborators on a message-routing problem domain in which the task was to identify how to get a message through an office network to recipients who were identified by their properties, e.g., "everyone involved in marketing for project two." The system was set up with 20 problem spaces defining parts of the overall problem. An external database held a large collection of problem instances involving different properties of the potential recipients and their interconnections. The Dispatcher problems were forced to be unique, i.e., they were generated randomly but selected from the problem domain without replacement. This was done to intentionally force the learning of a large number of chunks. Doorenbos' goal was to develop a chunk-matching algorithm whose overhead scaled better as a function of the number of chunks in the system. In the initial experiments, 10,000 chunks were learned (Doorenbos, Tambe & Newell, 1992). Dispatcher-Soar was later driven to create 113,938 chunks by working on 6,550 problems (Doorenbos 1993). He also exercised Soar in six other domains learning over 100,000 chunks in each (Doorenbos 1995) and demonstrated improvements in the match time by approximately two orders of magnitude.

Although identifying the characteristics of long-term learning was not the goal of Doorenbos' experiments, it is possible to reanalyze his data toward this end. By viewing his message routing experiments as long-term learning experiments involving 20 different problem spaces, we found interesting behavior.

Although a relatively constant chunking rate per problem was seen through the 6,550 problems, chunking in the 20 separate problem spaces varied widely. Chunking in nine of the problem spaces stopped at different points during the 6,550 problem solving session ranging from four stopping within the first 100 problems to four stopping after about 6,000 problems. Two more problem spaces were trending toward stopping after working all 6,550 problems. Another seven problem spaces tended toward a

constant number of chunks per problem, also stabilizing over the full range of problem solving. Finally, two problem spaces generated no chunks at all.

3.2 New Experimental Results

New experiments investigating the duration and rate of learning were conducted using the physics, Blocks-World, SCA, and simple and sophisticated tank domains. Analysis of the chunking rate in the physics problem space showed a clear downward trend through 400 problems.

In experiments with a range of Blocks-World problems, learning clearly did not go on forever. Experiments were run using five sets of 500 problems randomly generated with replacement. When running two-block problems, all problems were solved without learning any chunks. (With two blocks there are only three configurations and therefore only nine possible problems.) In experiments with three-block problems (169 possible problems), learning ended well before 150 random problems were attempted and with four-block problems, the longest run before learning ended was less than 100 problems. The results are shown in Table 1.

Table 1. End of learning in BlocksWorld Domains

Number of Blocks	Possible Problems	Problem at which Chunking Ended
2	9	No chunking done
3	169	79, 139, 33, 11, 36
4	5,329	76, 13, 69, 73, 17

In the Symbolic Concept Acquisition (SCA) system, chunking continued throughout the 500 training examples. This was by design because SCA adds a new chunk for each training example and repeated examples result in more specific chunks which the system preferred in making its classification. In experiments with a simple learning tank, all learning ended within 200 Soar decision cycles. For the sophisticated, planning tank, learning continued through 100,000 tank actions although the learning rate was decreasing as it had with the physics problem space.

3.3 Discussion

Two different behaviors were observed. The re-analysis of Doorenbos' data and the new experiments with the Blocks-World problem spaces demonstrated that learning can end even before all problems could have been seen due to the generality of Soar's learned chunks. However, it was also observed that learning can stabilize to a constant rate when the uniqueness of the problem instances was being enforced. We think this is a feature of problem spaces involving perpetual novelty.

Whether learning goes on forever depends in part on whether the problem space, i.e., the set of possible problems, is finite or infinite. For small, finite problem spaces, the intuitive answer is that learning will not go on forever because eventually every problem will be seen and solved at the very least by memorization. For a large finite and infinite problem space, the learning challenge is whether a system can learn things of a fairly general nature early enough to effectively solve any problem in the problem space, or "solve the space" without having seen and solved every possible problem instance in the problem space. In either case, there may be pragmatic reasons for monitoring the effectiveness of learning, and placing some controls on it (for example, Holder's suggestion that learning should be terminated based on recognition of degraded problem-solving time (Holder 1990).

4. How Much of Learned Knowledge Is Used?

Learning is based the expectation of future use of the knowledge gained. The validity of that expectation over long-term learning has not been investigated for symbolic machine learning. A discussion of how much of Soar's chunks are actually used is not found in the literature. Discussions with other Soar researchers revealed that many researchers do not operate Soar with learning turned on because the focus of their efforts is to model a characteristic of problem solving behavior and chunking makes it difficult to understand Soar's behavior.

A possible set of distinct patterns in the use of learned knowledge is based on when the rules were learned: (1) rules that are learned early on are used more frequently than rules learned later; (2) rules learned later (i.e., more recently) are used more frequently; and (3) rule use is independent of when a rule was learned. If rule use is independent of when it was learned, there would be no way to distinguish which rules should be kept based on when it was learned. Problem-solving time should then be related directly only to number of rules retained, i.e., memory size. If early learning is the primary basis of later solutions, then training sets would be more important than continuous learning. In this case, problem-solving performance would level off and not be significantly affected by either experience beyond the training set or by the size of memory larger than that necessary to hold the initial learning. On the other hand, if only the most recently learned rules are used, problem-solving time could tend to improve with experience as the early learning is replaced with later learning. In this case, the number of rules maintained above some threshold should not significantly affect problem-solving performance and the threshold for the necessary memory size could be established operationally.

4.1 Previous Work

Selective retention is the effect of the removal of learned rules based on low demonstrated value in problem solving, i.e., low utility. In a early paper on robot plans (Fikes, Hart, & Nilsson 1972), the authors recognized that continuing to learn plans would cause the problem solver to be faced with the "danger of being swamped by the ever-increasing repertoire of stored plans." They stated that the straightforward approach would be to keep statistics of the frequency with which plans were used and then discard those that fell below a specified threshold, but that could mask the dynamic nature of rule use. Variations on this approach were implemented but the utility problem remained.

Steve Minton's PRODIGY used a two-stage method to evaluate the utility of new control rules (Minton 1988). An estimate of the utility of proposed rules was used to initially decide whether they were worth saving. At the time of creation of a new rule, PRODIGY used the training example to estimate the costs and savings. If a rule represented a net savings, it was tentatively kept. Later measurements were to be used to decide whether to continue to keep the rule. In the reported problem space, of the 69 control rules estimated to be useful, only 19 were found to be useful based on measured use (Minton 1990).

Glenn Iba's MACLEARN (1989) also used a two stage utility filter to decide which macros were worth keeping. The initial, static filter performed a heuristic analysis of a proposed macro. The filter had three tests. The first was a check for redundancy with previous macros. The second was an effective-length threshold used to eliminate long macros. Finally, a domain-dependent test of specific features could be included. If these tests were passed, the macro was added to the knowledge base. Later, after completion of a training set of problems, Iba manually removed macros based on statistics of the macros' use. Credit for use was only given to the longest macros in the solution, not to sub-sequences within the longest macros. In practice, one use of a macro was enough to pass the manual dynamic filter.

Because little information was available on how much of the actual use of learned knowledge, new analysis was needed specifically to monitor chunk use.

4.2 New Results

Analysis of the previous experiments conducted using Soar addressed chunk use in simple physics problems, Blocks-World domains, SCA, and both the simple and sophisticated tanks.

One measure of the use of learning is a count of the number of chunks used throughout a long-term problem-solving run. Table 2 presents counts of overall chunk use for the physics, SCA, and sophisticated tank domains.

Table 2. Counts of Chunk Use

Count	Physics	SCA	TankSoar
Chunks saved	1,373	500	4,437
Chunks used at least once	756	228	2,591
Chunks used at least twice	689	143	1,864
Chunks used at least thrice	186	107	1,662
Chunks used more than 10x	31	66	1,130

Note that only about half of all chunks are ever used (55%, 46%, and 58%) and that a quarter or less are used more than 10 times (2%, 13%, and 25%).

Another measure of chunk use is the number of chunks used in solving each problem. In the physics domain, a near constant number of chunks were used per problem throughout the long run, as shown in Figure 1. However, closer examination of the chunk use revealed that which chunks were used was not at all constant. Figure 2 is one of a series of plots that indicate which chunks were used to solve 20 problems during the 400 problem solving run.

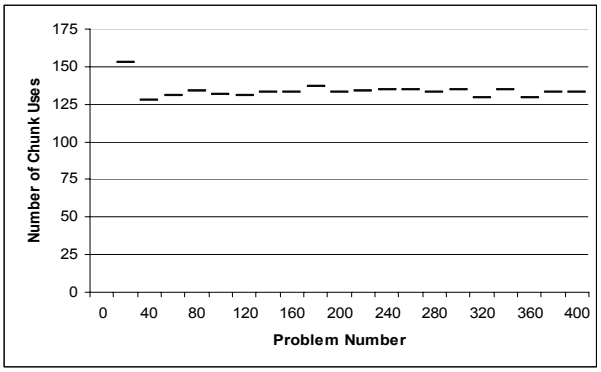

Figure 1. Physics chunk use over 400 problems

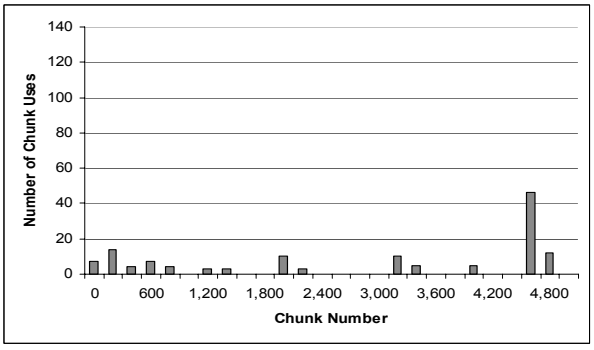

Figure 2. Physics chunk use on problems 381-400

Three observations are evident from review of plots like Figure 2 throughout the 400-problem run. First, while which chunks were used ranged over the entire set of chunks, the majority of the chunks used were those just created. The most recently learned chunks show the most uses because of the look-ahead approach used to evaluate alternative paths during problem solving. The look-ahead generates new chunks, which then they fire when the associated alternative is selected. The second observation is that chunks learned during the whole history of the problem solving were used. The experimental protocol of generating problems at random caused some repetition of previous conditions fitting the preconditions of previous chunks. Third, the reason an overall relatively constant number of chunks observed in Figure 1 is due to the increasing number of older chunks used and the decreasing number of new chunks used.

Experiments with three and four Blocks-World problems yielded similar average chunk use results. However, no patterns of chunk use over time were as clearly observed.

A third measure of chunk use is the monitoring of the use of individual chunks throughout the long-term problem solving experience. Figures 3 and 4 show the individual chunk uses for single runs in the 4-block Blocks-World and sophisticated TankSoar domains.

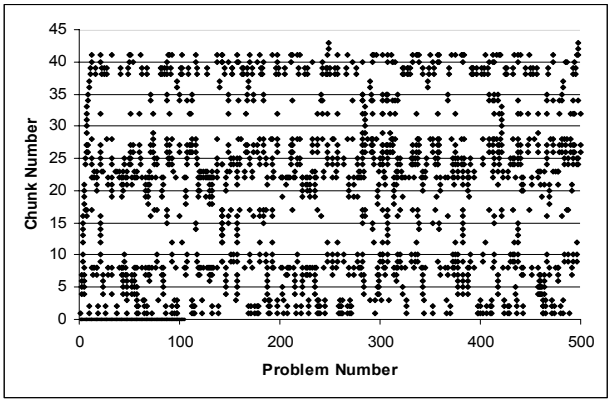

Figure 3. Individual chunk use in Blocks-World

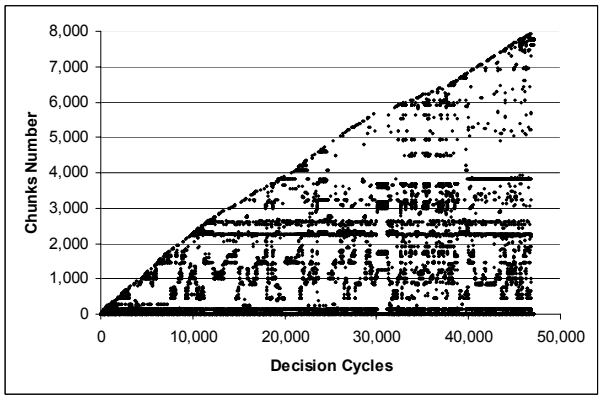

Figure 4. Individual chunk use by the sophisticated tank

Not only can frequently and infrequently used chunks be seen in Figures 3 and 4, but changes in chunk use throughout the runs can be seen. This shows an interaction between chunks learned at different points in the problem-solving experience.

4.3 Discussion

The question being investigated dealt with how much of the learned chunks were used. For the physics, SCA, and sophisticated tank domains, approximately half of the chunks generated were used. For the three-block Blocks-World problems, all 13 chunks were used during the five 500-problem runs.

For the physics problems, the clear pattern of use was that the most recently learned chunks made up the bulk of the chunk use, but this problem space continued learning over the entire 400 problem solving experience. During the much shorter learning phase in the Blocks-World domains, the higher use of more recently learned chunks may be observable, but it may simply be the effect of the very small set of chunks used. After learning ended, the use of chunks did not show a pattern based on when the chunks were learned.

4.4 Significance

Previous approaches to the utility problem have not exploited the actual use of the learned knowledge. Previous researchers have focused on the cost of learning and the cost of matching and projected. The only use information involved in previous work was whether or not the knowledge had been used at all (Iba 1989; Minton 1990). By understanding the nature of the use of learned knowledge, it should be possible to develop an approach to the utility problem based on actual, rather than projected behavior.

This work also has implications for a relationship between the size of memory and problem-solving time. The amount of learned knowledge retained for a specified problem solving performance level (in problem-solving time and possibly other performance measures) may be less than the complete retention currently used in many learning systems including Soar.

5. Addressing the utility problem

We now turn to using the characteristics of long-term learning to address the utility problem. Because a primary driver of the degraded performance problem-solving time is the number of rules in the system, several candidate approaches to excising chunks were considered.

The benefits of random forgetting were reported by Markovitch and Scott (1988). This approach uses information contained only in the first measure of the use of learned knowledge, that only a fraction is ever used.

Excising based on checking for use shortly after the learning was effectively implemented by Minton (1988) and Iba (1989), but that approach does not use the dynamic characteristics of long-term learning. Fikes, Hart, and Nilsson (1972) suggested keeping track of the least used knowledge. This approach also does not use the dynamic characteristics of long-term learning. Gratch and DeJong (1992) extended Minton's Prodigy by considering the interactions between rules in evaluating their usefulness, but they did not evaluate more data on actual rule use. Other approaches considered include removing the least recently used chunks (possibly to remain below a memory size limit) and removing chunks based on a gap between uses. Maintaining the necessary data on the use of each chunk to track least recently used chunks was expected to be computationally prohibitive although it may fit psychological models. We chose to excise chunks based on the gap between uses.

5.1 Projected Impact of Excising based on Use Gaps

Using the data on chunk use, it was possible to project the impact of excising chunks for different gap sizes. We analyzed when each chunk was used and plotting the number of uses that would be missed after the specified maximum gap in the domain studied. Figures 5 and 6 show a common transitional curve.

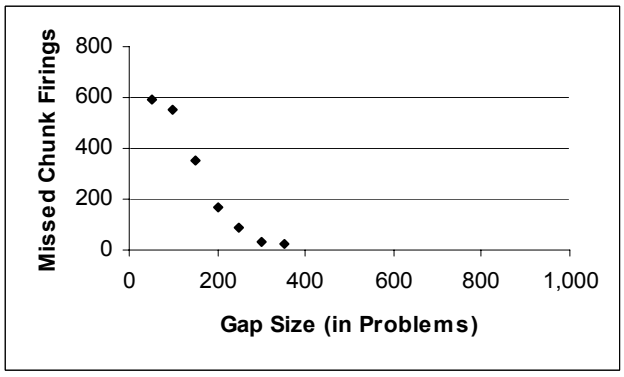

Figure 5. Impact of excising chunks in SCA domain

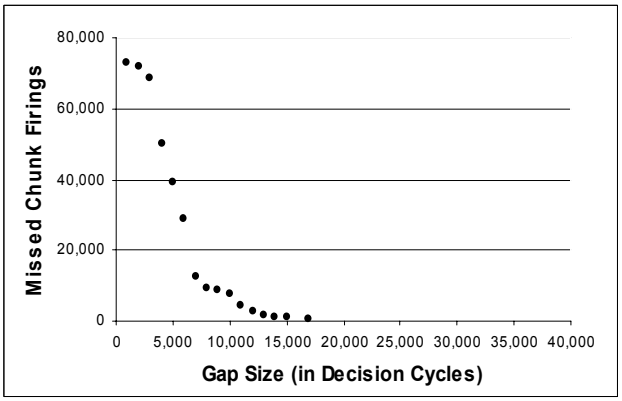

Figure 6. Impact of excising chunks in TankSoar

These figures show another characteristic of chunk use over the long term, a transition in the use of chunks with the gap between uses. This is useful because it can be exploited to excise low-use chunks.

The transition phenomenon was not studied fully to find a correlation to the problem domain, chunk development, or other factors. Its existence alone was sufficient to proceed to testing the idea of excising chunks to address the utility problem.

5.2 Excising Chunks in Soar based on Gap Size

Excising chunks in Soar based on the characteristics of chunk use is not easy. Although Soar was intentionally designed to not provide access to introspective data by the reasoning mechanisms, the user has access to such data through the user interface. The user interface also allows analysis of chunk use and excising of chunks based on that analysis. Details on how we got Soar to excise chunks based on a gap between chunk uses is described elsewhere (Kennedy 2002).

Figure 7 presents the number of chunks in the sophisticated TankSoar system with and without excising based on a gap of 5,000 decision cycles between uses.

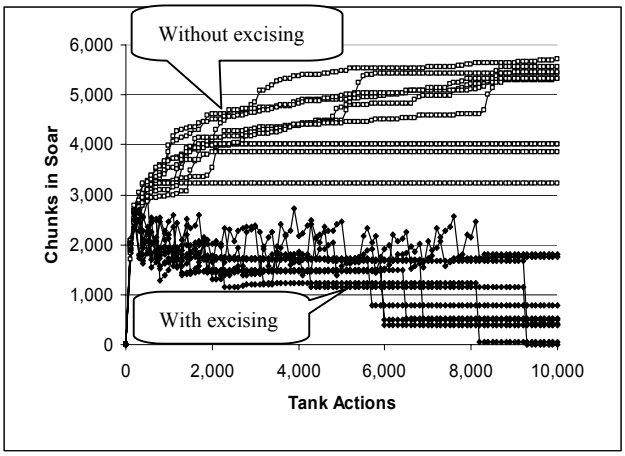

Figure 7. Excising chunks in TankSoar (same seed)

TankSoar has two sources of random behaviors, one in the TankSoar environment and one within Soar. The random behavior in the environment was addressed through the use of seeds for the random number generator. The random behavior caused by Soar itself was addressed by analyzing multiple runs.

The effect, in problem-solving time, of the same ten runs with and without excising chunks is shown in Figure 8. The mean and standard deviations are shown for the ten runs with the same seed for the environmental random variable and the same code with and without excising being implemented. For TankSoar, deciding which action the tank should take next was considered a problem. This figure shows that the cumulative time to solve a long

series of problems was statically significantly lower with excising implemented than without (using a t-test with a confidence factor of 99 percent).

Note that we are excising only learned chunks and not the basic domain knowledge the system is started with. Therefore, the problem-solving capability of the system is unaffected.

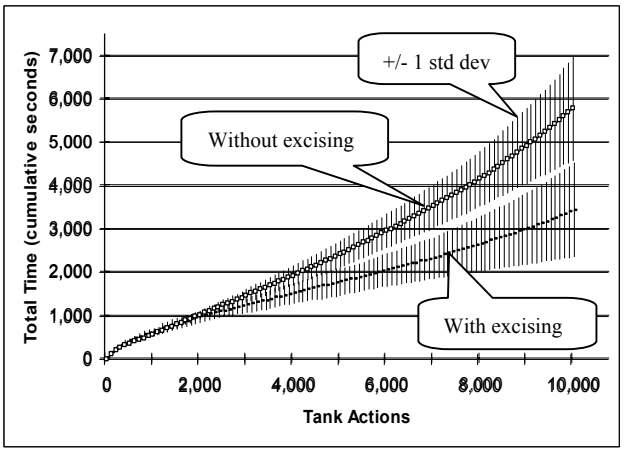

Figure 8. Cumulative problem-solving time with & without excising (same seed)

6. Conclusions and Further Research

This work is the first characterization of the behavior of long-term, on-line learning in Soar. We found was that learning does not go on forever (finite domains) and can end before all unique problems have been seen (based on the generality Soar's learned chunks). We also found that Soar's chunking can settle to a constant rate when novelty is guaranteed or forced (large or infinite domains).

On the use of learned knowledge, we found that many (as much as a half) of Soar's chunks are never used and although the overall chunking rate may be relatively constant, which chunks are used is not. Soar tends to use the most recently learned chunks more frequently, but also uses chunks from over the whole range of learning experience. (We also found some instances where previously useful learned knowledge can become useless because it has been replaced and postulated, but did not find, cases where learned knowledge is subsumed.)

The understanding of these characteristics of long-term learning in Soar allowed us to develop an approach to the utility problem based on excising low-use chunks. A characteristic curve based on the gap between chunk uses provided a basis for setting a threshold for excising low-use chunks. The improvement in Soar's problem-solving time, comparing with and without excising, was statistically significant.

Hopefully, these results will invigorate research into on-line learning. This work suggests a number of areas of future research In Machine Learning and in Cognitive Science.

Specific to Soar, the learning of low-use chunks, patterns in the use of chunks, and a more complete understanding of the transition in missed uses based on gap sizes need more thorough investigation. The modification of Soar to support forgetting, i.e., to support rule analysis and rule-based excising, is under consideration for a future version of Soar.

Beyond Soar, we hope other symbolic learning systems will be investigated to see if their long-term learning behavior has similar characteristics indicating fundamental characteristics of symbolic learning systems. In another direction, research into problem representations may be able to convert what are normally viewed as large problem spaces into effectively small ones. Finally, Allen Newell's premise concerning the retention of all knowledge as basic part of Soar as a Unified Theory of Cognition needs to be reconsidered. Forgetting may be necessary to overcome the utility problem and may be a fundamental part of cognition.

Acknowledgements

The authors would like to acknowledge the Bob Doorenbos who provided the raw data from his first excursion into long-term learning with Soar; Craig Miller who made his SCA system available; the Soar Group for their welcome and shared experiences; and certainly John Laird who leads the Group, maintains the culture of academic openness and sharing, and who volunteered the result of his most difficult Soar programming effort, the very sophisticated, planning TankSoar bot. The authors would also like to thank the ICML 2003 reviewers who provided many suggestions for improvements.

References

Blake, C., Keogh, E., & Merz, C.J. (1998). UCI Repository of machine learning databases (http://www.ics.uci.edu/~mlearn/MLRepository.html). Department of Information and Computer Science, University of California, Irvine, CA.

Bostrom, H. 1992. Eliminating Redundancy in Explanation-Based Learning. *Machine Learning: Proceedings of the Ninth International Workshop* (pp 37-42). San Mateo: Morgan Kaufmann.

Doorenbos, R. (1993). Matching 100,000 Learned Rules, *Proceedings of the Eleventh National Conference on Artificial Intelligence* (pp 290-296). Menlo Park: AAAI Press.

Doorenbos, R. B. (1995). *Production Matching for Large Learning Systems*, Doctoral Dissertation, Computer Science Department, Carnegie Mellon Univ.

Doorenbos, R., Tambe, T., & Newell, A. (1992). Learning 10,000 Chunks: What's it Like Out There?, *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp 830-836). Menlo Park: AAAI Press/The MIT Press.

Fikes, R.E., Hart, P.E., & Nilsson, N.J. (1972). Learning and executing generalized robot plans, *Artificial Intelligence* 3:251-288.

Gratch, J., & DeJong, G. (1992). COMPOSER: A probabilistic solution to the utility problem in speed-up learning. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp 235-240). San Jose: AAAI Press.

Holder, L. B. (1990). The General Utility Problem in Machine Learning, *Proceedings of the Seventh International Conference on Machine Learning* (pp 402-410). San Mateo: Morgan Kaufmann.

Iba, G. A. (1989). A heuristic approach to the discovery of macro operators, *Machine Learning* 3:285-317.

Kennedy, W. G. (2002). *Long-Term Learning in Soar and its Application to the Utility Problem*, Doctoral Dissertation, School of Information Technology and Engineering, George Mason Univ.

Korf, R. E. (1985). Macro-Operators: A Weak Method for Learning, *Artificial Intelligence* 26:35-77.

Laird, J. E., Congdon, C. B., Altmann, E., & Doorenbos, R. (1993). *Soar Users Manual*, Version 6, The Soar Group, School of Computer Science, Carnegie Mellon Univ.

Markovitch, S. & Scott, P.D. (1988). The role of forgetting in learning. *Proceedings of the Fifth International Conference on Machine Learning* (pp 459-465). Morgan Kaufmann.

Minton, S. (1988). *Learning Effective Search Control Knowledge: An Explanation-based Approach*, Doctoral Dissertation, Department of Computer Science, Carnegie Mellon Univ.

Minton, S. (1990). "Quantitative results concerning the utility of explanation-based learning," *Artificial Intelligence* 42:363-392.

Mooney, R. (1989). The Effect of Rule Use on the Utility of Explanation-Based Learning. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp 725-730). Los Altos: Morgan Kaufmann.

Newell, A. (1990). *Unified theories of cognition*. Cambridge: Harvard University Press.

Newell, A. (1993). "Allen Newell, Interviewed by B. Crandrasekaran," *IEEE Expert, June 1993: 5-12.*

Rosenbloom, P. & Newell, A. (1986). The Chunking of Goal Hierarchies, A Generalized Model of Practice. In *Machine Learning*, Vol. 2, R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.) Los Altos: Morgan Kaufmann.

Tambe, M., Newell, A., & Rosenbloom, P.S. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, V: 299-348.